Longest Common Subsequence Over Constant-Sized Alphabets: Beating the Naive Approximation Ratio

by

Shyan Akmal

B.S., Harvey Mudd College (2019)

Submitted to the Department of Electrical Engineering and Computer Science

in partial fulfillment of the requirements for the degree of

Masters of Science in Electrical Engineering and Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

September 2021

Author .....
Department of Electrical Engineering and Computer Science
August 24, 2021

Certified by .....
Virginia Vassilevska Williams
Steven and Renee Finn Career Development Associate Professor
Thesis Supervisor

Certified by .....
Ryan Williams
Professor of Electrical Engineering and Computer Science
Thesis Supervisor

Accepted by .....
Leslie A. Kolodziejski
Professor of Electrical Engineering and Computer Science
Chair, Department Committee on Graduate Students

Longest Common Subsequence Over Constant-Sized Alphabets: Beating the Naive Approximation Ratio

by

Shyan Akmal

Submitted to the Department of Electrical Engineering and Computer Science
on August 24, 2021, in partial fulfillment of the
requirements for the degree of
Masters of Science in Electrical Engineering and Computer Science

Abstract

This thesis investigates the approximability of the Longest Common Subsequence (LCS) problem. The fastest known algorithm for solving the LCS problem runs in essentially quadratic time in the length of the input, and it is known that under the Strong Exponential Time Hypothesis there can be no polynomial improvement over this quadratic running time. No similar limitation holds however, for approximate computation of the LCS, except in certain restricted scenarios. When the two input strings come from an alphabet of size k, returning the subsequence formed by the most frequent symbol occurring in both strings achieves a $1/k$ approximation for the LCS. It is an open problem whether a better than $1/k$ approximation can be achieved in truly subquadratic time ($O(n^{2-\delta})$ time for constant $\delta > 0$).

A recent result [Rubinstein and Song SODA'2020] shows that a $1/2 + \epsilon$ approximation for the LCS over a *binary* alphabet is possible in truly subquadratic time, provided the input strings have the same length. In this paper we show that if for some $\epsilon > 0$ a $1/2+\epsilon$ approximation is achievable for binary LCS in truly subquadratic time when the input strings can have differing lengths, then for every constant k there exists some $\delta_k > 0$ such that there is a truly subquadratic time algorithm that achieves a $1/k + \delta_k$ approximation for k-ary alphabet LCS. Thus, we show that for constant-factor LCS approximation, the case of binary strings is in some sense the hardest case. We also show that for every constant k, if one is given two strings of *equal* length over a k-ary alphabet, one can obtain a $1/k + \epsilon$ approximation for some constant $\epsilon > 0$ in truly subquadratic time. This extends the Rubinstein and Song result to all alphabets of constant size, and gives the first nontrivial improvement over the naive $1/k$ approximation for the LCS of strings over alphabets of size k for all $k \geq 3$.

Thesis Supervisor: Virginia Vassilevska Williams
Title: Steven and Renee Finn Career Development Associate Professor

Thesis Supervisor: Ryan Williams
Title: Professor of Electrical Engineering and Computer Science

Acknowledgments

First, I give my sincere and heartfelt thanks to Virginia Vassilevska Williams and Ryan Williams for being excellent advisors throughout my time at MIT thus far. They've both provided me with excellent guidance, engagement, and encouragement, and have helped me cultivate a deeper appreciation for many topics in theoretical computer science. Virginia and Ryan have been instrumental in helping me discover the topics in computer science and math I'm most passionate about, and have offered me phenomenal advice throughout my graduate studies. There have been times I've felt quite unsure about whether I'm cut out for research, but then after a quick conversation with Virginia or Ryan I feel refreshed, excited, and ready to tackle any problem that comes my way.

Next, I thank my fellow CS theory graduate students for exposing me to so many beautiful areas of computer science and math that I wouldn't otherwise know about, and helping make my experience at MIT fantastic. I give special thanks to Yinzhan Xu for reminding me that an SM thesis is something I should be submitting sooner rather than later, and for helping me get started with the process of writing this thesis. I thank Abhijit Mudigonda for no reason whatsoever, and I thank Tahsin Saffat for numerous reasons, none of which shall be revealed until the year 2025.

Finally, I thank my parents, Sayeed Akmal and Nahid Farhana, for all the love and support they've blessed me with, and for letting me stay with them throughout much of the COVID-19 pandemic. I also thank my younger brothers Ryaan Akmal and Zayan Akmal for occasionally distracting me from research, yet also patiently listening when I go on rants about complexity theory.

The results presented in this thesis are based on a collaboration with Virginia Vassilevska Williams.

Contents

Chapter 1

Introduction

1.1 Background & Motivation

Many scientific tasks involve measuring the similarity between two datasets. In a diverse array of applications, ranging from automated spell-checking to alignment of DNA sequences, the data we wish to compare is naturally represented as a string of letters. In this context, two of the the most foundational and popular measures of string similarity are the *longest common subsequence (LCS)* and the *edit distance*.

A "subsequence" of a string is a sequence of letters appearing left to right in the original string (for example, "truth" is a subsequence of "trustworthy"). Hence, given two strings A and B, their LCS (as the name suggests) simply measures the maximum length of a string which is a subsequence for both A and B. Intuitively, two strings with a larger LCS value are more similar than two strings with a smaller LCS.

In contrast, the edit distance measures how far apart two strings are by counting the minimum number of insertions, deletions and substitutions of characters that must be performed on one string to transform it into the other. These two measures are closely related: it turns out that the complement of the LCS is equal to the version of edit distance in which one minimizes only the number of insertions and deletions, while not allowing substitutions. So two strings with a larger edit distance value are considered less similar than two strings with a smaller edit distance.

Both the LCS and the edit distance of two length n strings can be computed exactly in $O(n^2)$ time using a classic dynamic programming approach. The fastest known algorithm for both problems was developed over 40 years ago by Masek and Paterson, and takes asymptotically $O(n^2/(\log n)^2)$ time in the worst case [MP80]. Unfortunately, for many applications (such as those related to genome sequencing) the relevant string length n is so large that this near quadratic running time is infeasible.

Moreover, it seems unlikely that substantially faster algorithms can be found: hardness results from fine-grained complexity have shown that *truly subquadratic time* algorithms (algorithms running in time $O(n^{2-\delta})$ time for some constant $\delta > 0$) for computing the LCS and edit distance cannot exist under the Strong Exponential

Time Hypothesis [ABV15, BI18, BK18] and other even more believable hypotheses concerning the complexity of circuit satisfiability problems [AHVW16, CGL+19].

Consequently, much recent research on sequence similarity has involved finding faster algorithms for returning good approximations for the LCS and edit distance metrics. For edit distance, there has been a long line of work obtaining better and better approximation algorithms, with the most recent result being a constant-factor approximation algorithm running in near-linear time [AN20].

In contrast, much less is known about how well the LCS can be approximated in truly subquadratic time. In fact, algorithms getting constant approximations for the LCS are only known over alphabets of constant size. When the input strings come from an alphabet of size k for some positive integer k, there is a naive $1/k$-approximation algorithm (meaning an algorithm that returns a value which is at least $1/k$ times the true LCS value) that returns in linear time the longest common *unary* subsequence of the two inputs. This algorithm finds this longest unary subsequence by simply going through each character in the alphabet and counting the maximum number of times it appears in both input strings.

Amazingly, despite the simplicity of this algorithm, it actually remains the best known LCS approximation algorithm that applies for all pairs of strings. Moreover, until very recently, no better constant-factor approximation algorithm was known for *any* constant size alphabet. In 2020, for the case of *binary* strings (i.e. strings over an alphabet of size two) of *equal* length, Rubinstein and Song [RS20] presented a breakthrough result demonstrating that it is possible to improve upon the naive 1/2-approximation algorithm described above. In particular, they obtained, for some constants $\epsilon, \delta > 0$, an $O(n^{2-\delta})$ time algorithm that returns a $(1/2+\epsilon)$-approximation for the LCS of two equal length binary strings.

This result motivates two natural questions.

First, is it possible to obtain such a $(1/2+\epsilon)$-approximation algorithm for binary strings of unequal length? Doing so would lead to the first approximation algorithm that truly beats the guarantee provided by the naive approximation algorithm over all pairs of binary strings.

Second, is it possible to extend this result and obtain a truly subquadratic time algorithm achieving a better than $1/k$-approximation for strings over an alphabet of size k, for every constant k? It is of great interest to get improved approximations beyond binary alphabets, since the natural alphabet size for data in applications may be much larger than two (for example, in biological applications we may be dealing with strings over an alphabet $\{\mathsf{A}, \mathsf{C}, \mathsf{G}, \mathsf{T}\}$ of size four corresponding to the bases occurring in DNA, and in applications involving human written text we may be dealing with strings over the English alphabet consisting of 26 letters). Rubinstein and Song went beyond the approximation ratio of 1/2 for LCS over binary strings – can we more generally break through the naive barrier of $1/k$ for the approximation ratio of LCS over alphabets of size k when $k > 2$?

1.2 Our Results

In this thesis we prove two results which make progress in addressing the questions raised in Section 1.1. The first result shows that if one can obtain a truly subquadratic time, better than 1/2-approximation algorithm for the LCS of binary strings of *possibly unequal lengths*, then one can use this algorithm to obtain a truly subquadratic time, better than $1/k$-approximation algorithm for the LCS of strings over a k-ary alphabet, for any constant k.

Theorem 1.1. *For any fixed integer $k \geq 2$ there is an $O(n)$ time algorithm that given an instance of LCS for strings of length at most n over an alphabet of size k, reduces it to $O(k^2)$ instances of LCS over* binary *strings of length at most n, so that $(1/2+\epsilon)$-approximate solutions (for $\epsilon > 0$) for these LCS instances can be translated in $O(n)$ time into a $(1/k+\epsilon_k)$ approximation of the k-ary alphabet LCS instance, where $\epsilon_k > 0$ is a constant only depending on ϵ and k.*

Said another way, in order to beat the longstanding naive $1/k$-approximation algorithm for LCS, one merely needs to obtain a better than 1/2 approximation for the binary case, i.e. to extend the Rubinstein and Song result to strings of possibly unequal length. This highlights the importance of the equal length restriction: the task of answering the first question raised at the end of Section 1.1 affirmatively is at least as hard as resolving the second question from the end of Section 1.1.

Our second result generalizes the Rubinstein-Song result by proving that one can beat the simple $1/k$-approximation algorithm for every constant k, as long as the input strings have equal length.

Theorem 1.2. *Given two strings A and B of length n over an arbitrary alphabet of size k, there exist positive constants ϵ and δ such that we can compute a $1/k+\epsilon$ approximation for the longest common subsequence of A and B in $O(n^{2-\delta})$ time.*

This theorem makes major progress on the second question raised at the end of Section 1.1, by showing how to extend Rubinstein and Song's result to hold not just for binary strings, but for strings from any constant size alphabet. In fact, our algorithm $(1/k+\epsilon)$-approximates the LCS for strings over an alphabet of size k in not just truly subquadratic time, but in *near-linear* $n^{1+o(1)}$ time.

Taken together, Theorem 1.1 and Theorem 1.2 show that it is possible to beat the naive approximation ratio for LCS over all constant size alphabets on equal length strings, and that to improve on existing approximations for all strings (even those of differing length) it suffices to obtain this improvement for binary strings.

Bibliographic Note The results and proofs presented in this thesis are based off work done in collaboration with Virginia Vassilevska Williams, and were previously published in the paper "Improved Approximation for Longest Common Subsequence over Small Alphabets" [AVW21b], which appeared in ICALP 2021.

1.3 Previous Work

To help place our results in the context of previous work, in this section we briefly survey the relevant literature on approximations for string similarity.

LCS Approximation We consider strings of length at most n over an alphabet Σ, and approximations to the LCS running in truly subquadratic time. As mentioned in Section 1.1, there is a naive $1/|\Sigma|$ approximation for the LCS of such strings that runs in linear time. This is a folklore result with no specific reference, but due to its simplicity has been known in the research community for essentially as long as LCS approximation has been studied. Prior to the results presented in this thesis, the only strict improvement over this trivial approximation algorithm applied in the case where the input strings have length n and $|\Sigma| = 2$, in which case there is a $1/2 + \epsilon$ approximation for the LCS [RS20].

If we assume the inputs satisfy some nice structural conditions, then it is possible to get additional interesting approximation algorithms for the LCS. For example, [RSSS19, Corollary II.3] shows that if the strings are chosen such that there is a character appearing in at least a $1/|\Sigma|$-fraction of the positions for both strings (i.e. intuitively, there is a common letter which appears quite frequently in both input strings), then it is possible to get a $\Omega(1/|\Sigma|^{3/4})$ approximation for the LCS. The constant hidden in the $\Omega(\cdot)$ notation here is quite small, so this gives an improvement over the naive algorithm only when $|\Sigma|$ is a very large constant. For general strings, no better approximation ratios are currently known than those already mentioned.

If we are willing to settle for subconstant approximation factors, then there has been interesting progress in the past few years on obtaining nontrivial LCS approximation algorithms in this regime. These results yield approximation ratios which do not depend on the size of Σ, and are thus interesting, for example, when the alphabet size is allowed to grows polynomially with the input, so that $|\Sigma| = \Omega(n^\delta)$ for some constant $\delta > 0$. A folklore sampling result in this area shows that for any $\rho \in (0, 1]$, there is an algorithm running in $O(n^{2-\rho})$ time which gives an $\Omega(n^{-\rho/2})$ approximation for the LCS. In particular, by setting $\rho = 1$ we see that we can $\Omega(n^{-0.5})$-approximate the LCS in $O(n)$ time. Recently, [HSSS19] obtained the first improvement over these sampling methods, by presenting an $O(n)$-time algorithm that obtains an $\Omega(n^{-(0.5-\epsilon)})$ approximation for some small positive $\epsilon \approx 0.002$. Soon after, [BD21] advanced on both of these results by showing that for any $\rho \in (0, 1]$, there is an algorithm running in $O(n^{2-\rho})$ time which (up to polylogarithmic factors) gives an $\Omega(n^{-2\rho/5})$ approximation for the LCS.

Edit Distance Approximation We consider strings of length at most n, and approximations to the edit distance running in truly subquadratic time. Unlike above, we write approximation ratios as larger than 1 (see the end of Section 1.4 for details about this convention). Research into edit distance approximation has had, at least so far, significantly more success than LCS approximation. Moreover, the ap-

proximation ratios obtained do not depend on the alphabet size. The first nontrivial approximation algorithm for edit distance was obtained over twenty years ago, and gave a $\sqrt{n}$ approximation in linear time [LMS98]. Since then a long line of work [BJKK04, BES06, AO12, AKO10] continued to obtain better and better approximation ratios, all of which, however, were still superconstant. Then in 2018, [BEG+18] presented a quantum algorithm which gave a constant-factor approximation to edit distance. All this research then culminated in the breakthrough result of [CDG+18], which showed how to replace the quantum aspects of the previous algorithm with randomized subroutines, and gave the first constant-factor approximation to edit distance running in truly subquadratic time. Several improvements, on both the runtime and concrete value of the approximation ratio, followed this breakthrough [And19, KS20, BR20]. The current state-of-the-art is given by Andoni and Nosatzki, who show that for any constant $\epsilon > 0$, we can compute a $(3+\epsilon)$ approximation for the edit distance in near-linear $n^{1+o(1)}$ time [AN20]. It is unclear in general how to get an approximation factor below 3, and it seems unlikely that existing methods can reduce this approximation factor much further [RW19, Section 4.3].

Fine-Grained Lower Bounds The Strong Exponential Time Hypothesis (SETH) asserts that any algorithm for solving CNF-SAT on n variables requires $2^{n-o(n)}$ time. This is a popular conjecture used frequently to prove conditional hardness results in fine-grained complexity, some of which can be found in the references within [Wil15, Section 2.3]. Refuting SETH would imply new *circuit lower bounds*, thereby resolving a major open problem in circuit complexity [Wil13, Wil14, JMV18]. It is known that assuming SETH, there is no $O(n^{2-\delta})$ time algorithm for computing the LCS or edit distance for any constant $\delta > 0$ [BI18]. If one believes SETH, then this is strong evidence that there is no truly subquadratic time algorithm for computing the LCS or edit distance. If one is skeptical of SETH, this result still shows a "circuit lower bound barrier" for getting faster exact algorithms for the LCS and edit distance, since, for example, getting an $O(n^{1.99})$ time algorithm for these problems would resolve a longstanding problem in circuit complexity, considered by the research community to be very difficult.

There have been several subsequent works showing even stronger hardness results. For example, [BK15] shows that this quadratic lower bound for LCS holds even when strings are binary, and [ABV15, BK18] give similar lower bounds for a wide variety of sequence similarity problems and different parameterizations for LCS. Quite recently, [AVW21a] proved that this quadratic lower bound can hold for LCS even when one of the input strings is fixed beforehand. In fact, assuming conjectures much more plausible than SETH, the work of [AHVW16] proves that LCS cannot have a truly subquadratic time algorithms, and that even obtaining $O(n^2/(\log n)^c)$ time algorithms for sufficiently large constants c would yield new circuit lower bounds.

These hardness results for exact computation motivate looking at approximation algorithms for the purpose of getting faster approaches for quantifying string and sequence similarity in practice.

When the inputs come from superconstant-sized alphabets, some fine-grained hardness results [AB17, AR18, CGL+19] are known for LCS approximation. However, these results generally rule out approximation ratios of the form $1 + o(1)$, so that the known lower bounds remain far from the best LCS approximation algorithms. Establishing stronger lower bounds for LCS approximation remains a very interesting open problem in fine-grained complexity.

Applications Although obtaining better LCS approximation algorithms is of great interest to theoretical computer science (due to the current scarcity of approximation algorithms in this area, as well as the strange gap between our knowledge of LCS and edit distance approximation), it also has potential applications in computational biology. For example, the heuristic algorithms presented in [BAFA16] suggest that better LCS approximation could potentially lead to better algorithms for genomic compression and phylogenetic reconstruction. More generally, LCS approximation is used as a subroutine in computational tasks across many scientific disciplines. For a survey of these applications and additional references, see [Nav01, Section 2].

1.4 Preliminaries

Notation Given a string A from an alphabet Σ, for any symbol $\sigma \in \Sigma$ we let $\sigma(A)$ denote the number of times σ appears in A. We say $\sigma(A)$ is the frequency of σ in A. When we refer to the LCS of A and B, we will sometimes refer to the string which is the longest common subsequence of A and B, and other times to the length of this string (this will be clear from context). Given strings A and B, we let $\mathrm{LCS}(A, B)$ denote the length of the longest common subsequence of A and B. Similarly, we let $\mathrm{ED}(A, B)$ denote the edit distance between A and B.

Useful Definitions In our reductions relating LCS approximation over different-sized alphabets, it will be helpful to have the following two definitions regarding string transformations and character frequencies.

Definition 1.3 (Restrictions)**.** *Given an alphabet Σ, we call a subset $\Sigma' \subseteq \Sigma$ a* subalphabet*. Given a string A from alphabet Σ, the* restriction *of A to a subalphabet Σ' is the maximum subsequence of A whose characters are all in Σ'.*

For example, if we fix the alphabet $\Sigma = \{\mathrm{h}, \mathrm{o}, \mathrm{r}, \mathrm{s}, \mathrm{t}, \mathrm{u}, \mathrm{w}, \mathrm{y}\}$ and consider the subalphabet $\Sigma' = \{\mathrm{h}, \mathrm{r}, \mathrm{s}, \mathrm{t}\}$, then the restriction of the "trustworthy" (a string from the alphabet Σ) to Σ' is "trstth."

Definition 1.4 (Balanced Strings)**.** *Given a string A of length n from an alphabet of size s and a parameter $\rho > 0$, we say a string is* ρ-balanced *if all its character frequencies are within ρn of n/s.*

For example, if we view "banana" as a string of length $n = 6$ over the alphabet $\{\text{a}, \text{b}, \text{n}\}$ of size $s = 3$, then we can verify that "banana" is 1/6-balanced because each of the letters "a", "b", and "n" appears at least $n/s - n/6 = 6/3 - 6/6 = 1$ time and appears at most $n/s + n/6 = 6/3 + 6/6 = 3$ times. However, "banana" is not 1/7-balanced because $n/s + n/7 = 6/3 + 6/7 < 3$ and "a" appears 3 times.

Edit Distance In this thesis, we focus on the version of edit distance where we only perform insertions and deletions to get from one string to another (in particular, we do not allow substitutions of characters). This version of edit distance and the standard edit distance metric with substitutions are always within a factor of 2 of one another. Thus, for the purpose of constant factor approximation algorithms these variants of edit distance are equivalent.

Approximation Ratios Since LCS is a maximization problem (we are tasked with finding the *longest* common subsequence), we write its approximation ratios as constants less than 1. A ρ-approximation for the LCS of A and B refers to a common subsequence of A and B whose length is at least ρ times the true LCS. So, for example, a $1/2 + \epsilon$ approximation algorithm for the LCS of A and B is an algorithm that returns a common subsequence of A and B with length at least $(1/2 + \epsilon) \cdot \text{LCS}(A, B)$.

It will not come up as frequently, but since edit distance is a minimization problem (we are tasked with finding the *fewest* number of insertions and deletions needed to turn one input string into the other), we write its approximation ratios as constants greater than 1. So, a c-approximation for the edit distance of A and B refers to a sequence of insertions and deletions which turn A into B, such that the number of insertions and deletions used is at most c times the edit distance between A and B.

Naive Approximation Given two strings A and B over an alphabet Σ, the naive approximation algorithm for LCS works as follows. First, we scan through all the characters in the strings A and B to compute $\sigma(A)$ for each symbol $\sigma \in \Sigma$. Then, for every symbol $\sigma \in \Sigma$ we compute $\min(\sigma(A), \sigma(B))$. If we consider the string which simply consists of the symbol σ repeated this many times, by definition we get a common subsequence of A and B. The algorithm then returns the longest such unary subsequence it finds – the length of the returned string is

$$\max_{\sigma \in \Sigma} \min(\sigma(A), \sigma(B))$$

by the description above.

We claim this yields a $1/|\Sigma|$ approximation to the LCS of A and B. Indeed, consider the true LCS of A and B. Let $\sigma_{\max} \in \Sigma$ be the symbol which appears most frequently in the LCS. By definition, $\sigma_{\max}$ accounts for at least a $1/|\Sigma|$ fraction of all characters in the LCS. However, since the LCS is a common subsequence of A and B, the symbol $\sigma_{\max}$ can appear at most $\min(\sigma_{\max}(A), \sigma_{\max}(B))$ times in the LCS. It

follows that the common subsequence returned by the algorithm above has length at least a $1/|\Sigma|$ fraction of the true LCS length.

Throughout the rest of this thesis, we generally refer to this algorithm as the "longest common unary subsequence approximation algorithm." On occasion, we may also informally refer to this algorithm as the trivial, naive, or counting approximation for the LCS.

1.5 Organization

In Chapter 2 we prove Theorem 1.1, and discuss why existing methods seem to require input strings to have equal length in order to get better than naive approximations to the LCS. In Chapter 3, we prove Theorem 1.2, conditioned on a special lemma about LCS approximation algorithms for highly structured strings. In Chapter 4 we engage in a thorough case analysis to prove the aforementioned special lemma which is required for the proof of Theorem 1.2. Finally, we end in Chapter 5 by presenting some concluding thoughts and interesting open problems in light of our results.

Chapter 2

The Significance of Equal Length

2.1 Reductions to Smaller Alphabets

In this chapter our goal is to prove Theorem 1.1 by showing how to reduce nontrivial constant-factor approximations of LCS over large alphabets to better than 1/2 approximations of LCS over binary alphabets. Although we do not directly apply this reduction in our proof of Theorem 1.2, the structure of this proof motivates the approach we end up using. Moreover, our reduction works even for strings of differing lengths, thus showing that one merely needs to extend the Rubinstein-Song result to unequal length strings in order to truly improve upon the naive approximation algorithm.

The proof is based off the idea of alphabet restrictions, introduced in Definition 1.3. We state our initial result as a general reduction between LCS approximation from larger alphabets to smaller alphabets.

Theorem 2.1. *Fix integers s and ℓ with $s > \ell \geq 2$. Suppose that there is a $T(n, \ell)$ time algorithm that achieves a $1/(\ell - \epsilon)$-approximation of the LCS of two strings of length at most n from an alphabet of size ℓ. Then, there is also a $O((n + T(n, \ell))\binom{s}{\ell})$ time algorithm that achieves an $1/(s(1-\epsilon/\ell))$-approximation of the LCS of two strings of length at most n from an alphabet of size s.*

Proof. We will show how to reduce the LCS for two strings of length at most n over an s-ary alphabet, to the LCS for two strings of length at most n over an ℓ-ary alphabet for any $\ell < s$. The reduction runs in $O(n\binom{s}{\ell})$ time and produces $\binom{s}{\ell}$ instances of ℓ-ary alphabet LCS.

Let A and B be two strings of length at most n over an alphabet Σ of size s. Then define C to be the longest common subsequence of A and B (note that we do not know the identity of C). For the sake of argument, sort the alphabet symbols according to their number of occurrences in C.

Let x be the collection of the ℓ most frequent alphabet symbols in C. Let C_x be the subsequence of C obtained by restricting C to the subalphabet of Σ that contains

the symbols of x. Since x has the ℓ most frequent symbols in C, C_x contains at least an ℓ/s fraction of the characters in C.

Now, let us describe our algorithm. Given A and B, we consider all subsets of the alphabet consisting of precisely ℓ symbols. Note that one of these subsets will be x. For each such collection y, consider the sub-instance of the LCS problem formed by restricting to the symbols of y. Let $\mathrm{OPT}(y)$ be the optimal LCS for this instance.

By definition, we have that

$$|\operatorname{OPT}(x)| \geq |C_x| \geq (\ell/s)|C|.$$

So when we consider $y = x$, if we can efficiently obtain an $1/(\ell - \epsilon)$ approximation for $\mathrm{OPT}(x)$, we will get a common subsequence of A and B of length at least

$$\frac{|\operatorname{OPT}(x)|}{\ell - \epsilon} \geq \frac{|C|}{s(1 - \epsilon/\ell)},$$

which yields the desired approximation for the LCS of A and B. The running time is multiplied by $\binom{s}{\ell}$, which is simply a constant provided s is bounded above by a constant. □

By setting $\epsilon = \ell\delta s/(1+\delta s)$ in the statement of Theorem 2.1, we obtain a linear time reduction from obtaining a $1/s + \delta$-approximation for s-ary strings to a $1/\ell + (\delta s)/\ell$-approximation for ℓ-ary strings. Thus, we derive Theorem 1.1 as a corollary:

Corollary 2.2 (Rephrasing of Theorem 1.1)**.** *Fix an integer $s \geq 3$ and a constant $\delta > 0$. The problem of obtaining a $1/s + \delta$ approximation for the LCS of two strings from an alphabet of size s can be reduced in linear time to the problem of obtaining a $1/2 + (\delta s)/2$ approximation for the LCS of two strings binary strings (i.e. strings from an alphabet of size two).*

Proof. This follows from Theorem 2.1 by taking $\ell = 2$. Note that the reduction has a multiplicative overhead of $O(s^2)$, which is constant when s is constant. □

Informally, Rubinstein and Song's algorithm gives a better than $1/2$ approximation for LCS over alphabets of size two. Our result above shows that an improvement over $1/2$ for the approximation ratio of binary LCS approximation implies an improvement over the ratio of $1/|\Sigma|$ for LCS approximation over any alphabet Σ of constant size. So, does combining the above result with the previous algorithm of [RS20] already yield better than $1/k$ approximation for equal length strings over alphabets of size k, thereby proving Theorem 1.2?

Unfortunately, this is not the case. The reason we cannot prove Theorem 1.2 by combining Corollary 2.2 with the result of [RS20] is that the latter gives a better than $1/2$ approximation for strings from an alphabet of size 2 only when the input strings have *equal length*. Note that in the reduction from the proof of Theorem 2.1, the

subsequences obtained from restrictions to subalphabets may be of different lengths, *even if the original strings have equal lengths.*

In Chapter 3 we will show how to get around this issue, essentially by designing a much more careful reduction on alphabet sizes that uses more information about the input strings (in particular, behaves differently depending on the the character frequencies of the input strings). Before showing that result, in the next section we offer some explanation for why having equal length input strings is amenable LCS approximation, at least when using existing techniques.

2.2 The Equal Length Barrier: Using Approximate Edit Distance

Although at first it may seem that extending the Rubinstein-Song result to strings of differing length should not be too hard (after all, there appears to be no such barrier in exact LCS computation, or approximate edit distance computation), generalizing the result has thus far resisted the research community's attempts. In the following we will discuss why this generalization may require very different approaches than those previously used. To see this, it will be helpful to call the following simple relationship between LCS and edit distance.

Lemma 2.3 (LCS and Edit Distance Connection)**.** *For any strings X and Y of length n and m respectively, we have*

$$2 \cdot \mathrm{LCS}(X, Y) + \mathrm{ED}(X, Y) = n + m.$$

For the sake of completeness, we include a proof of the above lemma.

Proof. Consider an optimal alignment between X and Y, which matches the maximum possible number of characters of X with identical characters of Y while respecting the order in which the characters appear in each string. The characters that are matched in X and Y correspond to a longest common subsequence. This is because if there were a longer common subsequence, we could get a larger alignment by matching the characters of that subsequence in X and Y, but this would contradict the optimality of our current alignment for X and Y.

Similarly, the unmatched characters correspond to a minimum set of symbols that need to be deleted from X and inserted from Y to turn X into Y. If there were a smaller edit distance computation, then all the characters which were not deleted or inserted could be paired up to form a larger alignment, again contradicting optimality.

Thus there are exactly $2 \cdot \mathrm{LCS}(X, Y)$ characters paired up in the alignment (since each of X and Y contributes $\mathrm{LCS}(X, Y)$ characters to the alignment) and $\mathrm{ED}(X, Y)$ unmatched characters. These encompass all the characters in X and Y, and thus account for $n + m$ symbols. □

In order to get an approximation ratio beyond 1/2 for the case of binary strings, [RS20] employs the following algorithm that can be viewed as a reduction from certain instances of LCS approximation to edit distance approximation. The reduction is motivated by the relationship identified in Lemma 2.3.

Definition 2.4 (Approximating LCS through Edit Distance)**.** *Given strings A and B of length n, the algorithm* ApproxED(A, B) *approximates the edit-distance between A and B and then returns the lower bound on the LCS implied by this. More precisely, the algorithm computes an approximate edit distance* $\widetilde{\mathrm{ED}}(A, B)$ *and then returns*

$$n - \frac{1}{2} \cdot \widetilde{\mathrm{ED}}(A, B). \tag{2.1}$$

As stated ApproxED can use any edit distance approximation $\widetilde{\mathrm{ED}}$ as a black box. For concreteness, we will take $\widetilde{\mathrm{ED}}$ to be the edit distance algorithm from [AN20] which can achieve an approximation ratio of c for any constant $c > 3$ and runs in near-linear time.

In [RS20], the authors use the ApproxED algorithm to handle the case of binary strings with large LCS. In this case, they notice that if the LCS is large then the edit distance must be small. Hence in this scenario, constant-factor approximations to edit distance will give good lower bounds on the LCS because in the calculation from Equation (2.1) we are subtracting off a small quantity from the maximum possible LCS value of n.

However, if we tried extending this algorithm to the case where the inputs X and Y have lengths n and m with $n = 100m$ (for example) by using the identity from Lemma 2.3, then even when the LCS is large (say of length $(1 - \epsilon)m$ for some small positive ϵ) the edit distance will still be very large compared to the length of the smaller string (at least $(99 + \epsilon)m$). So even a 3-approximation to edit-distance would incur massive error when trying to approximate LCS by computing

$$\frac{1}{2} \cdot \left(n + m - \widetilde{\mathrm{ED}}(X, Y) \right)$$

and the result would not give any nontrivial lower bound for the LCS.

To summarize, when the input strings have very different lengths it is not clear how to use approximate edit distance in general to obtain good approximations for LCS. This is essentially why the algorithm from [RS20] and Theorem 1.2 both require equal length inputs. To get better approximation algorithms for strings of differing length, it seems that either one would need to use some method completely different from edit distance approximation, or one would need to find more intricate reductions to edit distance approximation that leverage the structure of the string more intelligently.

Chapter 3

Better Approximation Beyond Binary Alphabets

The goal of this section is to prove Theorem 1.2, thereby beating the naive $1/|\Sigma|$ approximation ratio for all alphabets Σ of constant size. For convenience, we restate the theorem with slightly different wording below.

Theorem 3.1 (Rephrasing of Theorem 1.2)**.** *Given two strings A and B of length n over an arbitrary alphabet Σ of size s, there exist a positive constant ϵ such that we can compute a $1/s+\epsilon$ approximation for the longest common subsequence $\mathrm{LCS}(A,B)$ of A and B in truly subquadratic time.*

Throughout the rest of this section, we assume A and B refer to strings satisfying the conditions of Theorem 3.1, and that $s \geq 3$. We begin by establishing lemmas corresponding to easy instances of the problem. To describe these instances, we will appeal to the notion of "balanced" strings, introduced previously in Definition 1.4.

Lemma 3.2 (Balanced Inputs, adapted from Lemma 3.2 of [RS20])**.** *For all sufficiently small $\rho > 0$, if either A or B is ρ-balanced, we can $(1/s+\gamma)$-approximate $\mathrm{LCS}(A,B)$ in truly subquadratic time, where γ is some positive constant depending on ρ.*

Proof. We prove this result by showing how to reduce LCS approximation between a string and a balanced string to edit distance approximation.

Without loss of generality assume A is ρ-balanced. This means that all of its character frequencies are at least $(1/s-\rho)n$. Hence, there exists a unary common subsequence of A and B of at least this length. If this common subsequence acts as a $(1/s+\gamma)$ approximation to the LCS we are done.

If this is not the case, the LCS must be quite large, in the sense that

$$\mathrm{LCS}(A,B) > s(1/s-\rho)n - s\gamma n = (1-s(\rho+\gamma))n.$$

Now, recall from Lemma 2.3 that

$$\mathrm{ED}(A, B) + 2 \cdot \mathrm{LCS}(A, B) = 2n.$$

. Using the edit distance approximation of [AN20] with some approximation ratio $c = 3.1$ (for example) in the algorithm outlined in Definition 2.4, we recover an common subsequence between A and B of length at least

$$n - c\,(n - \mathrm{LCS}(A, B)) > n\,(1 - cs(\rho + \gamma))\,.$$

Provided we pick γ such that

$$1 - cs(\rho + \gamma) \geq 1/s + \gamma,$$

this common subsequence then works as a $(1/s + \gamma)$ approximation for the LCS as desired. This latter bound holds provided we have

$$\gamma \leq \frac{s - 1 - cs^2\rho}{s(1 + cs)}.$$

We can ensure the above inequality holds by picking ρ small enough in terms of c and s so that the numerator of the right hand side above is positive, and then setting the constant γ to be positive and smaller than the right hand side. Note that smaller values of ρ correspond to larger values of γ.

To summarize, the algorithm works by finding a long common unary subsequence and a common subsequence inferred derived from existing edit distance approximation algorithms. We then return the larger of these two strings, and the above reasoning shows that this output will be a $(1/s + \gamma)$-approximation to the LCS as desired. □

The next lemma shows how we can leverage not just frequency conditions in the input string, but frequency conditions in the LCS. This is one of the key observations that allows our algorithm to extend beyond the reach of the algorithm from [RS20].

Lemma 3.3 (Imbalanced LCS)**.** *If the LCS of A and B is not ρ-balanced, then in linear time we can $(1/s + \rho/(s-1))$-approximate* $\mathrm{LCS}(A, B)$.

Proof. It turns out that returning the longest common unary subsequence gives the desired approximation. To see this, let $\sigma_{\max}$ and $\sigma_{\min}$ be the most frequent and least frequent characters in the LCS respectively. Since the LCS is not balanced, it must be the case that either $\sigma_{\max}$ makes up more than a $(1/s + \rho)$ fraction of all symbols, or $\sigma_{\min}$ makes up fewer than a $(1/s - \rho)$ fraction of the symbols, in the LCS.

In the latter case, the $s - 1$ members of the alphabet besides $\sigma_{\min}$ must account for at least an $((s-1)/s) + \rho$ fraction of characters in the LCS. Among these, by definition $\sigma_{\max}$ appears the most often, which means by averaging that $\sigma_{\max}$ accounts

for at least a

$$\frac{1}{s} + \frac{\rho}{s-1}$$

fraction of all symbols in the LCS.

In the former case $\sigma_{\max}$ certainly satisfies this property as well. Thus, in either case we know that $\sigma_{\max}$ makes up at least a

$$\frac{1}{s} + \frac{\rho}{s-1}$$

fraction of the symbols in the LCS.

Then the string consisting of $\sigma_{\max}$ repeated $\min(\sigma_{\max}(A), \sigma_{\max}(B))$ times is a common subsequence of A and B which has at least as many instances of $\sigma_{\max}$ as the LCS does, and thus yields the desired approximation. □

So far in our goal of proving Theorem 3.1, we have identified certain easy cases in which it is easy to get a better than $1/s$ approximation for the LCS of equal length strings from an alphabet of size s.

To handle strings not falling into these cases, our next idea is to attempt to find restrictions of A and B to binary subalphabets and then invoke frequency arguments from previous work to obtain a better approximation. As we noted before, we cannot directly use the reduction in Corollary 2.2 because the better than $1/2$ approximation of [RS20] only applies when the inputs have the same length.

It turns out however, that the arguments of [RS20] *do* hold for strings of differing length as long as the inputs satisfy a particularly rigid frequency condition. The following lemma demonstrates how by careful choice of subalphabets we can find restrictions meeting this condition. This argument and the combinatorial observations used in the proof constitute one of our primary conceptual contributions to the previous work of [RS20] and are crucial in generalizing to larger alphabet sizes.

Lemma 3.4 (Binary Restriction)**.** *Suppose neither A nor B are ρ-balanced. Then there exists a subalphabet $\Sigma' \subseteq \Sigma$ with $|\Sigma'| = 2$ such that the restrictions of A and B to Σ' are not ρ/s-balanced.*

Proof. Let $\alpha_1 \leq \cdots \leq \alpha_s$ be the number of times the distinct symbols $\sigma_1, \ldots, \sigma_s$ of Σ appear respectively in A, so that σ_s is the most frequent symbol of A and σ_1 is the least common character in A. By averaging we know that $\alpha_s \geq n/s \geq \alpha_1$.

Since A is not ρ-balanced we deduce that

$$\alpha_s - \alpha_1 > \rho n.$$

We can decompose the left hand side of the above equation to a sum

$$\alpha_s - \alpha_1 = \sum_{i=2}^{s} (\alpha_i - \alpha_{i-1}) .$$

Since all the summands on the right hand side of the above equation are positive, there must exist an index j with the property that

$$\alpha_j - \alpha_{j-1} > (\rho/s)n. \tag{3.1}$$

Note that we can find such a j in linear time by scanning through A and B and keeping counts of all the characters that appear.

Now, consider the $(s-1)$ two-element sets

$$\{\sigma_s, \sigma_{j-1}\}, \{\sigma_{s-1}, \sigma_{j-1}\}, \ldots, \{\sigma_j, \sigma_{j-1}\}, \{\sigma_j, \sigma_{j-2}\}, \ldots, \{\sigma_j, \sigma_1\}. \tag{3.2}$$

We claim that one of these sets satisfies the conditions required of Σ' from the lemma statement.

First, note that Equation (3.1) ensures that the restriction of A to any of the above sets is not ρ/s-balanced, so it suffices to simplify verify that the restriction of B will not be balanced for one of these sets. Suppose to the contrary that none of the sets from Equation (3.2) satisfies the desired conditions. We use a triangle-inequality argument on the character frequencies of B to derive a contradiction.

Let M and m be indices such that σ_M is the most frequent character of B and σ_m is the least frequent. Since B is not ρ-balanced, we know that $M \neq m$.

If $M \geq j > m$ we may write

$$|\sigma_M(B) - \sigma_m(B)| \leq |\sigma_M(B) - \sigma_{j-1}(B)| + |\sigma_{j-1}(B) - \sigma_j(B)| + |\sigma_j(B) - \sigma_m(B)|.$$

By assumption, the restrictions of B to the sets $\{\sigma_M, \sigma_{j-1}\}$, $\{\sigma_j, \sigma_{j-1}\}$, and $\{\sigma_j, \sigma_m\}$ are each (ρ/s)-balanced. Thus, each addend on the right hand side of the above inequality is bounded above by $(\rho/s)n$. It follows that

$$|\sigma_M(B) - \sigma_m(B)| \leq (3\rho/s)n.$$

By similar reasoning, if $m = j$ and $M \geq j$ then we have

$$|\sigma_M(B) - \sigma_m(B)| = |\sigma_M(B) - \sigma_j(B)| \leq |\sigma_M(B) - \sigma_{j-1}(B)| + |\sigma_{j-1}(B) - \sigma_j(B)| \leq (2\rho/s)n.$$

If instead $M, m > j$, we can instead bound

$$|\sigma_M(B) - \sigma_m(B)| \leq |\sigma_M(B) - \sigma_{j-1}(B)| + |\sigma_{j-1}(B) - \sigma_m(B)| \leq (2\rho/s)n$$

since now the subalphabets $\{\sigma_M, \sigma_{j-1}\}$ and $\{\sigma_m, \sigma_{j-1}\}$ both occur in Equation (3.2). Similar reasoning on the remaining cases of the values of M and m relative to j to establishes the inequality

$$\sigma_M(B) - \sigma_m(B) \leq (3\rho/s)n$$

regardless of the particular values of m and M.

We have dropped absolute value signs because the left hand side of the above equation is positive by definition of M and m. Since $s \geq 3$ this contradicts the assumption that B is not ρ-balanced, and the desired result follows. Note that we can find which of subalphabets from Equation (3.2) satisfies the conditions of the lemma in $O(n + s)$ time by scanning through B to get the counts of each of its characters and trying out all the restrictions. □

The purpose of the above lemma is to reduce LCS approximation on large alphabets to LCS approximation over binary alphabets, for strings with some particular structural properties. The next lemma then shows how to get a better than $1/2$ approximation to the LCS for binary strings of possibly differing length, *provided* they satisfy some special frequency condition (which corresponds to the structural properties ensured by the Lemma 3.4).

Lemma 3.5. *Let X and Y be binary strings of length n and m respectively, where $m \leq n$. Suppose the frequencies $0(Y)$ and $1(X)$ are both at most $(1/2 - \rho)m$ for some positive constant ρ. Then there exist positive constants δ and ϵ such that if $0(Y)$ and $1(X)$ are within δm of each other, we can compute a $(1/2 + \epsilon)$ approximation of $\mathrm{LCS}(X, Y)$ in subquadratic time.*

The above result of Lemma 3.5 is essentially a generalization of a theorem proved by Rubinstein and Song [RS20, Section 4]. This variant of their theorem is useful because it applies to strings of different length. The proof is quite long, so we defer the justification of Lemma 3.5 to Chapter 4.

The main technique behind the proof is a casework on the character frequencies found in the two input binary strings. Most of the proof is identical to partition method of [RS20], although there are a few cases requiring modifications to the argument. Conceptually, the main contribution here is the realization that in certain limited cases, we can get around the equal length barrier discussed in Section 2.2 by partitioning strings in slightly different ways than previous work.

Before moving onto the proof Theorem 3.1, we prove one last lemma that leverages the character frequency information of the input strings to identify another easy case in LCS approximation.

Lemma 3.6 (Lemma 3.1 from [RS20])**.** *Let X and Y be binary strings. For any constant $\delta > 0$, if X and Y satisfy*

$$\min(0(X), 0(Y)) > (1 + \delta)\min(1(X), 1(Y))$$

or

$$\min(1(X), 1(Y)) > (1 + \delta)\min(0(X), 0(Y))$$

then there is a unary $(1 + \delta)/(2 + \delta)$ approximation for $\mathrm{LCS}(X, Y)$.

Proof. Observe that for any binary strings X and Y we have

$$\mathrm{LCS}(X, Y) \leq \min(0(X), 0(Y)) + \min(1(X), 1(Y)). \tag{3.3}$$

This equation holds because the LCS is a subsequence of X and Y, and thus cannot contain more zeros or ones than either of the strings X or Y individually have.

Suppose by symmetry that $\min(0(X), 0(Y))$ is the larger of the two addends on the right. Then the we can return the all zeros string of length $\min(0(X), 0(Y))$. By construction, this is a common subsequence of X and Y. Moreover, by the first inequality from the lemma statement we have

$$\min(0(X), 0(Y)) > (1+\delta)(\mathrm{LCS}(X,Y) - \min(0(X), 0(Y)))$$

and then rearranging proves the claim. □

We now illustrate how to combine all the results established thus far to obtain a better LCS approximation algorithm over all alphabets of constant size.

Proof of Theorem 3.1. Let ρ and ρ' be positive parameters (bounded below by a constant) whose values will be specified later.

If either of the strings A or B is ρs-balanced, we are done by Lemma 3.2 (by taking ρ to be small enough so that the lemma applies). If the LCS of A and B is not ρ'-balanced, we are done by Lemma 3.3. So, we may assume that neither A nor B are ρs-balanced and that their LCS is ρ'-balanced.

By Lemma 3.4 we can find binary alphabet restrictions X and Y of A and B respectively with the property that neither X nor Y are ρ-balanced. Informally, since the LCS of A and B is balanced, a better than $1/2$ approximation for the LCS of X and Y acts as a better than $1/s$ approximation for $\mathrm{LCS}(A, B)$.

More precisely, since $\mathrm{LCS}(A, B)$ is ρ'-balanced, each of its characters occurs at least $(1/s - \rho')\,\mathrm{LCS}(A, B)$ times in the LCS. By construction, we also know that $\mathrm{LCS}(X, Y)$ is at least as large as a binary alphabet restriction of $\mathrm{LCS}(A, B)$. Thus, it follows that given a positive constant ϵ', a $1/2 + \epsilon'$ approximation of $\mathrm{LCS}(X, Y)$ has length at least

$$(1/2 + \epsilon') \cdot 2\,(1/s - \rho')\,\mathrm{LCS}(A, B) = (1/s + 2\epsilon'/s - \rho' - 2\epsilon'\rho')\,\mathrm{LCS}(A, B).$$

By setting $\epsilon = \epsilon'/s$ (for example) and ρ' sufficiently small in terms of ϵ', the above calculation shows that a $1/2 + \epsilon'$ approximation for the LCS of X and Y acts as a $1/s + \epsilon$ approximation for $\mathrm{LCS}(A, B)$. Hence to complete the proof, it suffices to get a better than $1/2$ approximation for $\mathrm{LCS}(X, Y)$.

We may assume that

$$\min(0(X), 0(Y)) \leq (1+\delta)\min(1(X), 1(Y)) \tag{3.4}$$

and

$$\min(1(X), 1(Y)) \leq (1+\delta)\min(0(X), 0(Y)) \tag{3.5}$$

for some constant δ since otherwise Lemma 3.6 yields a better than $1/2$ approximation.

As mentioned previously, [RS20] gives a better than $1/2$ approximation for the LCS when X and Y have equal length. Hence, it suffices to consider the case where X and Y have different lengths. In this situation, by symmetry, we may assume without loss of generality that $|X| > |Y|$ and $0(Y) \leq 1(Y)$.

Our next step is to perform casework on the character frequencies of X, relative to the character frequencies of Y.

Since X is longer than Y, we cannot have both $1(X) \leq 1(Y)$ and $0(X) \leq 0(Y)$.

If $1(X) > 1(Y)$ and $0(X) > 0(Y)$ simultaneously, then eq. (3.5) implies that

$$1(Y) \leq (1+\delta) \cdot 0(Y)$$

which contradicts the fact that Y is not ρ-balanced as long as we take $\delta \leq 2\rho$.

If instead $1(X) \leq 1(Y)$ and $0(X) \leq 0(Y)$, by eq. (3.5) we similarly have

$$0(X) \leq 0(Y) \leq 1(Y) \leq (1+\delta) \cdot 0(X)$$

which again contradicts the fact that Y is not ρ-balanced for the same choice of δ.

Thus, the only possibility is that $1(X) \leq 1(Y)$ and $0(X) > 0(Y)$.

Let $m = |Y|$ be the length of string Y. Then eq. (3.4) implies that

$$0(Y) \leq (1+\delta) \cdot 1(X) \leq 1(X) + \delta m$$

while eq. (3.5) implies that

$$1(X) \leq (1+\delta) \cdot 0(Y) \leq 0(Y) + \delta m.$$

Consequently, $0(Y)$ and $1(X)$ are within δm of each other. Then by Lemma 3.5, as long as we take δ small enough in terms of ρ we get a subquadratic $1/2 + \epsilon'$ approximation for $\mathrm{LCS}(X, Y)$. As noted earlier, this then yields the desired $1/s + \epsilon$ approximation for $\mathrm{LCS}(A, B)$ in truly subquadratic time.

In fact, because we only ever use subroutines that run in linear time or constant factor approximations to edit distance which take near-linear time, the overall algorithm takes near-linear time. □

Chapter 4

Better Approximation for Imbalanced Strings

In Chapter 3 we proved Theorem 3.1, showing how to beat the naive $1/|\Sigma|$ approximation ratio for LCS approximation on equal length strings from any alphabet Σ of constant size. The proof was self-contained, except for the omission of the justification for a supporting result, Lemma 3.5, that was employed in the argument. This section is devoted to filling in that justification, by proving Lemma 3.5.

The proof is structured by working through the individual arguments in the case analysis of [RS20] and verifying that the arguments still hold in the case where the strings have different lengths, as long as they satisfy the frequency requirements included in the hypotheses of the lemma.

Throughout this chapter we fix the binary alphabet $\Sigma = \{0, 1\}$ and assume that all strings come from this alphabet. For ease of description, we carry over the following subroutines from [RS20].

Definition 4.1. *Given strings A and B and a symbol σ, the algorithm* $\mathrm{Match}(A, B, \sigma)$ *returns the largest subsequence of A and B consisting entirely of copies of σ.*

Definition 4.2. *Given strings A and B, the algorithm* $\mathrm{BestMatch}(A, B)$ *returns the longer of the strings*

$$\mathrm{Match}(A, B, 0) \quad \textit{and} \quad \mathrm{Match}(A, B, 1).$$

In other words, the algorithm returns the largest common unary subsequence.

Note that BestMatch is a $1/2$ approximation algorithm for LCS.

Definition 4.3. *Given strings A_1, A_2, and B, the algorithm* $\mathrm{Greedy}(A_1, A_2, B)$ *returns*

$$\max_{B=B_1 \sqcup B_2} \mathrm{BestMatch}(A_1, B_1) + \mathrm{BestMatch}(A_2, B_2)$$

where the maximum is taken over all possible splits of B into two contiguous left and right substrings B_1 and B_2.

These algorithms can all be implemented to run in linear time by scanning through the counting how many 0s and 1s appear in each string.

We will also utilize the procedure outlined in Definition 2.4 that infers an LCS approximation via a constant-factor approximation algorithm for computing the edit distance of two strings. As before, for concreteness we assume that the routine leverages the algorithm $\widetilde{\text{ED}}$ from [AN20], which runs in near-linear time and can approximate the edit distance to a $c = 3 + \epsilon'$ factor for any fixed positive constant ϵ'. Note that this algorithm also returns a common subsequence achieving the given length.

Bypassing the Equal Length Barrier In the statement of Lemma 3.5, the input strings are not required to have equal length. But as noted in Section 2.2, we only know how to use edit distance to get improved LCS approximations when the input strings are equal length. How do we overcome this obstacle?

The key idea is to apply edit distance approximation not to the full string, but carefully chosen substrings. If we identify substrings in the input strings of the same length, then we can use edit distance approximation on those equal length strings to get a good estimate for part of an approximate LCS. We can then use combinatorial arguments or other routines to get approximations for other parts of the LCS, and then combine it with the partial solution we found using edit distance to form a good approximation to the full LCS of the original strings.

For general strings of differing length, it still remains unclear how to identify these sorts of substrings whose approximate LCS can be extended to a good approximation for the full LCS. However, in the proof below, we manage to show that when we have strings meeting the frequency conditions from the statement of Lemma 3.5, it is possible to identify such substrings by considering frequencies of zeros and ones in the inputs. In particular, the strings under consideration are quite imbalanced, so we can either identify some balanced substrings (which is amenable to approximation by edit distance), or just find imbalanced substrings which have the same bias towards zeros or ones (in which case the longest unary subsequence algorithm will give a good approximation).

Proof of Lemma 3.5. Let $1(X) = \alpha m$ for some $\alpha \in [0, 1]$. By assumption, we know that $\alpha < 1/2$ is bounded away from $1/2$ by some constant amount. We can also assume that $0(Y)$ is within δm of $1(X)$ for some positive parameter $\delta < 0$ to be picked later on. We will end up setting δ to be some sufficiently small constant depending on α.

For the remainder of this proof, we use the notation $\approx$ to denote quantities that are within δm of each other. For example $1(X) \approx 0(Y)$. This lets us avoid having to stick in $\pm\delta m$ symbols in all the inequalities and helps make the arguments cleaner without affecting the correctness (since we just care about getting a $1/2 + \epsilon$ approximation for *some* constant $\epsilon > 0$).

Note that by Equation (3.3) the LCS of the input strings

$$\mathrm{LCS}(X, Y) \leq (2\alpha + \delta)m \tag{4.1}$$

cannot be too large, because the LCS contain at most αm 1s from X and $\approx \alpha m$ 0s from Y.

Define R_X and R_Y to be the substrings of X and Y consisting of their rightmost αm characters respectively. Similarly define L_X and L_Y as the substrings of X and Y consisting of their respective leftmost αm characters. Set $M_X = X \setminus (L_X \cup R_X)$ and $M_Y = Y \setminus (L_Y \cup R_Y)$ to be the middle substrings of X and Y that remain when these left and right ends are chopped off.

We now follow the casework of [RS20], explaining at each step why the arguments still hold in our more general setting. These cases are based off the frequencies of 0s and 1s in the left and right ends of the inputs, and consider separately the situation where these substrings are pseudorandom (balanced) or structured (imbalanced).

In the former case we can appeal to edit distance as in the proof of Lemma 3.2, and in the latter situation we can exploit the imbalance in the strings to use the simpler unary algorithms described in Definition 4.1, Definition 4.2, and Definition 4.3. Intuitively, we succeed in using edit distance approximation arguments (and overcome the barrier described the end of Section 2.2) in this particular case because even though X and Y may have different size, we identify right and left substrings which all have equal length and only employ edit distance approximation around these areas.

Recall that, by assumption, $\alpha < 1/2 - \rho$ for some constant ρ. We select a parameter $\beta < \rho/20$ to represent deviation from the balanced case. By choosing δ sufficiently small in terms of β we may additionally assume that

$$1(X), 0(Y) < (1/2 - 10\beta)\, m \tag{4.2}$$

since in the hypothesis of the lemma we supposed that $1(X) < (1/2 - \rho)m$ and that $0(Y) \approx 1(X)$.

Finally, as one last piece of notation in this proof, we write $|A|$ to denote length of an arbitrary string A. We now begin the casework, maintaining consistency with the previous casework argument of [RS20].

4.1 Case 1(a)

Case 1(a): $1(R_Y), 0(R_X) \in [(\alpha/2 - 4\beta)\, m, (\alpha/2 + 4\beta)\, m]$

In this case both right ends of the strings are balanced. We will give a good approximation for the LCS by splitting these strings at the right ends and using the aforementioned algorithms.

Consider an optimal alignment between X and Y (i.e. a maximum partial matching of identical characters in X and Y, corresponding to the LCS of X and Y). Now define $\widehat{R_Y}$ to be the minimal suffix (from the right end) of the string Y with the

property that every character from R_X which is matched in the alignment is paired up with some character in $\widehat{R_Y}$.

Without loss of generality, we may assume that $\widehat{R_Y}$ is a substring of R_Y. This is because if $\widehat{R_Y}$ was not contained in R_Y, we could define an analogous substring of $\widehat{R_X}$ of X satisfying $\widehat{R_X} \subseteq R_X$ and then use a symmetric argument to get the desired approximation.

Let $\widehat{L_Y} = Y \setminus \widehat{R_Y}$ be the left substring of Y that remains after chopping of $\widehat{R_Y}$.

Optimality of the alignment implies that

$$\mathrm{LCS}(X, Y) = \mathrm{LCS}(X \setminus R_X, \widehat{L_Y}) + \mathrm{LCS}(R_X, \widehat{R_Y}). \tag{4.3}$$

Define the quantities

$$f_L = \min(1(X \setminus R_X), 1(\widehat{L_Y})) + \min(0(X \setminus R_X), 0(\widehat{L_Y}))$$

and

$$f_R = \min(1(R_X), 1(\widehat{R_Y})) + \min(0(R_X), 0(\widehat{R_Y}))$$

which represent frequency-based upper bounds for the LCS terms from the right hand side of Equation (4.3). They are useful because Equation (3.3) together with Equation (4.3) implies that

$$\mathrm{LCS}(X, Y) \leq f_L + f_R. \tag{4.4}$$

We also introduce the quantity

$$Z = \max\left(\min(1(X \setminus R_X), 1(\widehat{L_Y})), \min(0(X \setminus R_X), 0(\widehat{L_Y}))\right)$$

which is the larger of the two addends defining f_L, and is equal to the length of the string returned by the subroutine

$$\mathrm{BestMatch}(X \setminus R_X, \widehat{L_Y}).$$

We further subdivide into cases based off the size of Z.

Case 1(a)(i): $Z > (\alpha/2 + 10\beta)\, m$

When Z is large we can combine two unary subsequences to get a good enough LCS approximation. We first show that Z is bigger than $f_L/2$ by a constant fraction of m.

By definition we have

$$f_L - Z = \min\left(\min(1(X \setminus R_X), 1(\widehat{L_Y})), \min(0(X \setminus R_X), 0(\widehat{L_Y}))\right) \leq 1(X \setminus R_X).$$

Since X has αm ones and R_X has length αm we get that

$$1(X \setminus R_X) = \alpha m - 1(R_X) = \alpha m - (\alpha m - 0(R_X)) = 0(R_X).$$

Then using the case assumptions we have

$$0(R_X) \leq (\alpha/2 + 4\beta)\, m < Z - 6\beta m.$$

Chaining these inequalities together and rearranging we deduce that

$$Z > f_L/2 + 3\beta m.$$

Now if we make a single call to the Greedy routine we obtain a string of length

$$\text{Greedy}(X \setminus R_X, R_X, Y) \geq \text{BestMatch}(X \setminus R_X, \widehat{L_Y}) + \text{BestMatch}(R_X, \widehat{R_Y}). \quad (4.5)$$

From our earlier discussion we have

$$\text{BestMatch}(X \setminus R_X, \widehat{L_Y}) \geq Z > f_L/2 + 3\beta m.$$

Moreover

$$\text{BestMatch}(R_X, \widehat{R_Y}) = \max(\min(1(R_X), 1(\widehat{R_Y})), \min(0(R_X), 0(\widehat{R_Y}))) \geq f_R/2$$

since we are taking the maximum over two addends that sum to f_R.

Finally, if we substitute the above inequalities into Equation (4.5) and apply Equation (4.4) we get that

$$\text{Greedy}(X \setminus R_X, R_X, Y) \geq (f_L + f_R)/2 + 3\beta m \geq \text{LCS}(X, Y)/2 + 3\beta m \geq (1/2 + 3\beta)\, \text{LCS}(X, Y).$$

Thus running the subroutine Greedy yields a better than 1/2 approximation in this case.

Case 1(a)(ii) : $Z \leq (\alpha/2 + 10\beta)\, m$

Intuitively, in this case Z is too small for us to ensure a good approximation using frequency guarantees alone. However, because Z is so small, the LCS also cannot be too large. Because of this, we will be able to get a good approximation by combining a common subsequence of the (balanced) right ends of the strings with a common subsequence of the strings with the right ends removed.

More concretely, we leverage the ApproxED algorithm from Definition 2.4. From

our previous observation we have

$$f_L \leq 2Z \leq (\alpha + 20\beta)\, m.$$

So via Equation (4.3) we can bound the LCS by

$$\mathrm{LCS}(X, Y) \leq f_L + \mathrm{LCS}(R_X, \widehat{R_Y}) \leq (\alpha + 20\beta)\, m + \mathrm{LCS}(R_X, R_Y) \tag{4.6}$$

where in the last step we also used the fact that $\widehat{R_Y} \subseteq R_Y$.

We will get an approximation by returning a unary subsequence from the left parts of the strings, and using edit distance approximation on the right ends. First, we can make a call to BestMatch and get a common subsequence of length at least

$$\mathrm{BestMatch}(X \setminus R_X, Y \setminus R_Y) \geq \mathrm{Match}(X \setminus R_X, Y \setminus R_Y, 0) \geq (\alpha/2 - 4\beta)\, m.$$

This last inequality above follows from combining the **case 1** assumptions about the frequencies of characters in $0(R_X)$ and $1(R_Y)$, together with the facts that

$$0(Y) \approx 1(X) = \alpha m$$

and

$$|R_X| = |R_Y| = \alpha m.$$

Now, since R_X and R_Y are $(4\beta/\alpha)$-balanced we can apply Lemma 3.2 with $n = \alpha m$ as long as we take β sufficiently small in terms of α. Note that for fixed alphabet size and ρ, the parameter β remains $\Omega(1)$. This ensures that in subquadratic time we can compute a common subsequence of R_X and R_Y with length at least

$$(1/2 + \gamma)\, \mathrm{LCS}(R_X, R_Y)$$

where $\gamma < 1$ is the constant from Lemma 3.2, which is larger for smaller values of β. By the frequency assumptions on R_X and R_Y, we know that

$$\mathrm{LCS}(R_X, R_Y) \geq (\alpha/2 - 4\beta)\, m$$

so in fact the subsequence of R_X and R_Y returned by Lemma 3.2 has length at least

$$(1/2 + \gamma)\, \mathrm{LCS}(R_X, R_Y) \geq \mathrm{LCS}(R_X, R_Y)/2 + \gamma(\alpha/2 - 4\beta)m.$$

Now if we combine these two subsequences together, we get a common subsequence of X and Y of length at least

$$\mathrm{LCS}(R_X, R_Y)/2 + (\alpha/2 - 4\beta)\, m + \gamma(\alpha/2 - 4\beta)m$$

which can be written in the form

$$\left((\alpha + 20\beta)m + \mathrm{LCS}(R_X, R_Y)\right)/2 + (\gamma\alpha/2 - 14\beta - 4\gamma\beta)\, m.$$

By applying Equation (4.6) and the fact that $\gamma < 1$ we see that this is at least

$$\mathrm{LCS}(X, Y)/2 + (\gamma\alpha/2 - 18\beta)\, m.$$

Finally, by picking β small enough this expression is at least

$$\mathrm{LCS}(X, Y)/2 + (\gamma\alpha/3)\, m \geq (1/2 + \gamma/6)\, \mathrm{LCS}(X, Y)$$

where in the last step we have used Equation (3.3) together with $1(X) = \alpha m$ and $0(Y) \approx \alpha m$. This proves that we can attain a better than $1/2$ approximation for the LCS as claimed.

4.2 Cases 1(b), 1(c), and 2

Case 1(b): $1(R_Y) < (\alpha/2 - 4\beta)\, m$ **and** $0(R_X) \leq (\alpha/2 + 2\beta)\, m$

In this case the right ends of X and Y each do not contain too much their respective strings' most common characters. We show that this implies the LCS of both strings must be so small that simply returning a unary string yields a better than $1/2$ approximation.

As in case 1(a), consider an optimal alignment between X and Y. Now define the substring $\widehat{R_Y}$ to be the minimal suffix of Y which contains all characters of Y that R_X is aligned to. Let $\widehat{L_Y} = Y \setminus \widehat{R_Y}$ be what remains of Y after R_Y is removed. Since the alignment is optimal, Equation (4.3) holds.

We now subdivide into further case based off how the right ends of strings are aligned.

Case 1(b)(i): every character of R_X is matched to some character of R_Y in the alignment.

In this case $\widehat{R_Y}$ is a substring of R_Y. From Equation (3.3) and the case assumptions we get that

$$\mathrm{LCS}(X \setminus R_X, \widehat{L_Y}) \leq 1(X \setminus R_X) + 0(\widehat{L_Y}) \leq (\alpha/2 + 2\beta)\, m + 0(\widehat{L_Y})$$

and

$$\mathrm{LCS}(R_X, \widehat{R_Y}) \leq 1(\widehat{R_Y}) + 0(\widehat{R_Y}) \leq (\alpha/2 - 4\beta)\, m + 0(\widehat{R_Y}).$$

Note that in this second inequality we are using the inequality $1(\widehat{R_Y}) \leq 1(R_Y)$

which follows from the earlier observation that $\widehat{R_Y} \subseteq R_Y$. By adding these inequalities together and substituting the result into Equation (4.3) we deduce that

$$\mathrm{LCS}(X,Y) \leq (\alpha - 2\beta)\, m + \left(0(\widehat{L_Y}) + 0(\widehat{R_Y})\right) \leq (\alpha - 2\beta)\, m + \alpha m = (2\alpha - 2\beta)\, m$$

where we have used the fact that the symbol 0 occurs in Y a total of $\approx \alpha m$ times.

Case 1(b)(ii): some character of R_X is matched outside R_Y in the alignment.

In this case R_Y is a substring of $\widehat{R_Y}$.

Using Equation (3.3) again we find that

$$\mathrm{LCS}(X \setminus R_X, \widehat{L_Y}) \leq 1(X \setminus R_X) + 0(\widehat{L_Y}) \tag{4.7}$$

From $0(R_X) \leq (\alpha/2 + 2\beta)\, m$ we know that $1(R_X) \geq (\alpha/2 - 2\beta)\, m$ since R_X has length αm. It follows that

$$1(X \setminus R_X) = 1(X) - 1(R_X) \leq (\alpha/2 + 2\beta)\, m$$

since $1(X) = \alpha m$.

Similarly, since $1(R_Y) < (\alpha/2 - 4\beta)\, m$ and R_Y has length αm we know that $0(R_Y) > (\alpha/2 + 4\beta)\, m$. Since $R_Y \subseteq \widehat{R_Y}$ it must be the case that $0(\widehat{R_Y}) > (\alpha/2 + 4\beta)\, m$ which means that

$$0(\widehat{L_Y}) = 0(Y) - 0(\widehat{R_Y}) \approx \alpha m - 0(\widehat{R_Y}) < (\alpha/2 - 4\beta)\, m.$$

Adding these two inequalities and substituting into Equation (4.7) proves that

$$\mathrm{LCS}(X \setminus R_X, \widehat{L_Y}) \leq (\alpha - 2\beta)\, m.$$

Then applying Equation (4.3) and using the fact that R_X has length αm proves

$$\mathrm{LCS}(X,Y) = \mathrm{LCS}(X \setminus R_X, \widehat{L_Y}) + \mathrm{LCS}(R_X, \widehat{R_Y}) \leq (\alpha - 2\beta)m + \alpha m = (2\alpha - 2\beta)\, m.$$

Thus in both this subcase and the previous one, the LCS is at most $(2\alpha - 2\beta)\, m$. Hence, returning the string of $\approx \alpha m$ zeros obtained by calling $\mathrm{Match}(X, Y, 0)$ gives a better than 1/2 approximation as desired.

Case 1(c): $1(R_Y) \leq (\alpha/2 + 2\beta)\, m$ **and** $0(R_X) < (\alpha/2 - 4\beta)\, m$

This case is symmetric to case 1(b) and similar reasoning handles it.

Case 2: $1(L_Y), 0(L_X) \leq (\alpha/2 + 2\beta)\, m$

Combining cases 1(a), 1(b), and 1(c) resolves the situation where

$$1(R_Y), 0(R_X) \leq \alpha/2 + 2\beta.$$

Consequently, case 2 is symmetric to case 1. In particular, we can flip the strings (by replacing X with the string X' which consists of the symbols of X read in reverse from right to left, and replacing Y with its analogous reverse string Y') and apply the arguments from case 1 to handle this case.

4.3 Cases 3 and 4

Case 3: $1(L_Y), 1(R_Y) \leq (\alpha/2 + \beta)\, m$ **and** $0(L_X), 0(R_X) > (\alpha/2 + 2\beta)\, m$

In this case both ends of the inputs have many 0s, so repeated calls to Match will be enough to guarantee a better than 1/2 approximation.

Since L_Y has length αm and at most $(\alpha/2 + \beta)\, m$ instances of 1, it must have at least

$$0(L_Y) \geq \alpha m - (\alpha/2 + \beta)m = (\alpha/2 - \beta)m$$

occurrences of 0. Since $1(R_Y) \leq (\alpha/2 + \beta)\, m$ the same reasoning shows that

$$0(R_Y) \geq (\alpha/2 - \beta)\, m.$$

Consequently we can get common subsequences of the left and right ends of the strings consisting entirely of 0s with lengths

$$\text{Match}(L_X, L_Y, 0) = \min(0(L_X), 0(L_Y)) \geq (\alpha/2 - \beta)\, m$$

and

$$\text{Match}(R_X, R_Y, 0) = \min(0(R_X), 0(R_Y)) \geq (\alpha/2 - \beta)\, m.$$

Intuitively, since the ends of the strings have many 0s, we expect the middle substrings to have many 1s. Indeed, since the left end

$$1(L_X) = |L_X| - 0(L_X) < \alpha m - (\alpha/2 + 2\beta)m = (\alpha/2 - 2\beta)\, m$$

of X does not have many 1s and similar reasoning shows that

$$1(R_X) < (\alpha/2 - 2\beta)\, m$$

this holds for the right end as well, the middle of X has at least

$$1(M_X) = 1(X) - 1(L_X) - 1(R_X) > \alpha m - 2(\alpha/2 - 2\beta)m = 4\beta m$$

appearances of 1.

To argue that M_Y has enough 1s we will need to appeal to the fact that Y is not balanced. From Equation (4.2) we know that

$$1(Y) = m - 0(Y) > m - (1/2 - 10\beta)m = (1/2 + 10\beta)\, m.$$

We can use the case assumptions together $\alpha < 1/2$ to then deduce

$$1(M_Y) = 1(Y) - 1(L_Y) - 1(R_Y) > (\alpha + 10\beta)m - 2(\alpha/2 + \beta)m = 8\beta m.$$

Consequently the largest all 1s subsequence between the middle substrings has length at least

$$\text{Match}(M_X, M_Y, 1) = \min(1(M_X), 1(M_Y)) \geq 4\beta m.$$

Thus, by combining these three subsequences from three calls to match we can recover in linear time a common subsequence of size

$$\text{Match}(L_X, L_Y, 0) + \text{Match}(M_X, M_Y, 1) + \text{Match}(R_X, R_Y, 0) > 2(\alpha/2 - \beta)m + 4\beta m = (\alpha + 2\beta)m.$$

By Equation (4.1) the true LCS has length at most $\approx 2\alpha m$, so a string of length $(\alpha + 2\beta)\, m$ is a $1/2 + \epsilon$ approximation for some constant $\epsilon < 2\beta/\alpha$ as desired.

Case 4: $0(L_X), 0(R_X) \leq (\alpha/2 + \beta)\, m$ **and** $1(L_Y), 1(R_Y) > (\alpha/2 + 2\beta)\, m$

This case is symmetric to case 3 and similar reasoning handles it.

4.4 Cases 5 and 6

Case 5: $0(L_X), 1(R_Y) > (\alpha/2 + \beta)\, m$

In this case, the ends of the strings have unusually large instances of either 0 or 1. This will enable us to combine two calls to Match to get the desired approximation.

We can check that the right end of Y does not have many 0s

$$0(R_Y) = |R_Y| - 1(R_Y) < \alpha m - (\alpha/2 + \beta)m = (\alpha/2 - \beta)\, m.$$

It follows that the remainder of the string $Y \setminus R_Y = L_Y \cup M_Y$ has many zeros

$$0(Y \setminus R_Y) = 0(Y) - 0(R_Y) \approx \alpha m - 0(R_Y) > (\alpha/2 + \beta)\, m.$$

Similar reasoning shows that

$$1(L_X) = \alpha m - 0(L_X) < (\alpha/2 - \beta)\, m$$

so that the remaining portion of X has many ones

$$1(X \setminus L_X) = 1(X) - 1(L_X) = \alpha m - 1(L_X) > (\alpha/2 + \beta)\, m.$$

It follows that

$$\text{Match}(L_X, Y \setminus R_Y, 0) + \text{Match}(X \setminus L_X, R_Y, 1) > (\alpha + 2\beta)\, m$$

using these match subroutines and combining the resulting strings yields a better than $1/2$ approximation, since by Equation (4.1) the LCS is at most $\approx 2\alpha m$.

Case 6: $1(L_Y), 0(R_X) > (\alpha/2 + \beta)\, m$ This is symmetric to case 5 and a similar argument proves the result holds in this situation.

By inspection or by referring to [RS20, Table 1], we can verify that these cases handle all possible input strings satisfying the conditions of the lemma. Since in every case we obtain a better than $1/2$ approximation for the LCS in subquadratic time (and in fact near-linear time for our choice of edit distance approximation algorithm), we have proven the claim. □

Chapter 5

Conclusions

5.1 Summary of Work

Given two input strings over an alphabet Σ of constant size, there is a counting algorithm that runs in linear time and computes a $1/|\Sigma|$ approximation to the longest common subsequence (LCS) of the inputs. It has been a longstanding open problem in the field of approximation algorithms to design algorithms which run in truly subquadratic time and beat this naive approximation ratio.

In this thesis we prove Theorem 1.2, which shows how beat this ratio and obtain better than $1/|\Sigma|$ approximations for the LCS of *equal-length strings* over any constant-sized alphabet Σ. This completely resolves the above open problem for equal length strings, and while a similar result had been known previously for binary strings [RS20], this is the first such improvement for all alphabets of size at least $|\Sigma| \geq 3$.

Additionally, we make partial progress towards eliminating the need for this equal length condition, by proving Theorem 1.1, which shows that to get better LCS approximation for strings of possibly differing length over all alphabets, it suffices to get better LCS approximation for binary strings of possibly differing length. We also show how to get this improved approximation ratio for certain classes of binary input strings of unequal length.

Philosophically, this result shows that for LCS approximation, the case of binary strings appears to be the hardest, and that removing the equal length condition is at least as hard as obtaining improved approximation algorithms in general for all constant alphabet sizes. Having broken the psychological barrier of a $1/|\Sigma|$ approximation ratio, it seems that it should be of great interest to see how much larger this approximation ratio for LCS can be made.

From a technical point of view, our work provides a new reduction for LCS approximation over alphabets of different sizes, shows a slightly more general fashion in which edit distance approximation can be used to improve LCS approximation, and develops new ways of leveraging frequency information of the input strings and their LCS in designing better approximation algorithms.

5.2 Open Problems and Future Directions

The main problem suggested by our work is of course: can we beat the naive approximation ratio for LCS even when the strings have different lengths? That is, given two strings of length at most n over an alphabet of size s, does there exist an algorithm which runs in truly subquadratic time and returns a $1/s + \epsilon$ approximation to the LCS of the input strings, for some constant $\epsilon > 0$? By Theorem 1.1, to answer this question it suffices to resolve it in the case of binary strings, where $s = 2$.

Another question raised by our work is to what extent can we increase the approximation ratios for LCS over an alphabet Σ above $1/|\Sigma|$? Our algorithm and that of Rubinstein and Song's only beat the naive approximation ratio by a modest amount. How large of an ϵ can we find while still having a truly subquadratic algorithm that computes a $1/|\Sigma| + \epsilon$ approximation? In our work, the value of ϵ also decays polynomially as $|\Sigma|$ grows. Can this dependence be improved?

Alternatively, can we prove better hardness results (conditioned on conjectures from fine-grained complexity, for example) for ruling out certain types of LCS approximation? The positive results in this thesis and other works such as [RSSS19] may be helpful for ruling out easy cases and identifying potential hard cases for LCS approximation.

To prove better algorithmic results, it could potentially be helpful to employ the ideas of edit distance approximation more explicitly instead of reducing to this problem in a black-box manner as we do in our work. For handling inputs of different length, perhaps there could be benefits to reducing to variants of edit distance which are asymmetric and involve approximating the distance to many substrings of one of the inputs (problems like these were used previously in [And19] as helper algorithms in edit distance approximation).

In our work, outside of using constant-factor edit distance approximation, all the algorithms we used were extremely simple and only involved counting character frequencies. In particular, although we exploit information about the order of characters implicitly in how we partition strings in Chapter 4, it seems like our algorithms are still agnostic to much of the linear structure of both the input strings and the true LCS. It could be the case that using more costly routines (which still run in truly subquadratic time, but not necessarily linear time) or more involved algorithmic paradigms which leverage information about the strings more explicitly could lead to even better algorithms.

Bibliography

[AB17] Amir Abboud and Arturs Backurs. Towards hardness of approximation for polynomial time problems. In *8th Innovations in Theoretical Computer Science Conference, ITCS 2017, January 9-11, 2017, Berkeley, CA, USA*, volume 67 of *LIPIcs*, pages 11:1–11:26, 2017. 14

[ABV15] Amir Abboud, Arturs Backurs, and Virginia Vassilevska Williams. Tight hardness results for LCS and other sequence similarity measures. In *IEEE 56th Annual Symposium on Foundations of Computer Science, FOCS 2015, Berkeley, CA, USA, 17-20 October, 2015*, pages 59–78. IEEE Computer Society, 2015. 10, 13

[AHVW16] Amir Abboud, Thomas Dueholm Hansen, Virginia Vassilevska Williams, and Ryan Williams. *Simulating Branching Programs with Edit Distance and Friends: Or: A Polylog Shaved is a Lower Bound Made*, page 375–388. Association for Computing Machinery, New York, NY, USA, 2016. 10, 13

[AKO10] Alexandr Andoni, Robert Krauthgamer, and Krzysztof Onak. Polylogarithmic approximation for edit distance and the asymmetric query complexity. In *51th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2010, October 23-26, 2010, Las Vegas, Nevada, USA*, pages 377–386. IEEE Computer Society, 2010. 13

[AN20] Alexandr Andoni and Negev Shekel Nosatzki. Edit distance in near-linear time: it's a constant factor. *CoRR*, abs/2005.07678, 2020. 10, 13, 20, 22, 30

[And19] Alex Andoni. Simpler constant-factor approximation to edit distance problems, 2019. 13, 42

[AO12] Alexandr Andoni and Krzysztof Onak. Approximating edit distance in near-linear time. *SIAM J. Comput.*, 41(6):1635–1648, 2012. 13

[AR18] Amir Abboud and Aviad Rubinstein. Fast and deterministic constant factor approximation algorithms for LCS imply new circuit lower bounds. In Anna R. Karlin, editor, *9th Innovations in Theoretical Computer Science Conference, ITCS 2018, January 11-14, 2018, Cambridge, MA,*

USA, volume 94 of *LIPIcs*, pages 35:1–35:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018. 14

[AVW21a] Amir Abboud and Virginia Vassilevska Williams. Fine-Grained Hardness for Edit Distance to a Fixed Sequence. In Nikhil Bansal, Emanuela Merelli, and James Worrell, editors, *48th International Colloquium on Automata, Languages, and Programming (ICALP 2021)*, volume 198 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 7:1–7:14, Dagstuhl, Germany, 2021. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. 13

[AVW21b] Shyan Akmal and Virginia Vassilevska Williams. Improved Approximation for Longest Common Subsequence over Small Alphabets. In Nikhil Bansal, Emanuela Merelli, and James Worrell, editors, *48th International Colloquium on Automata, Languages, and Programming (ICALP 2021)*, volume 198 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 13:1–13:18, Dagstuhl, Germany, 2021. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. 11

[BAFA16] Richard Beal, Tazin Afrin, Aliya Farheen, and Donald Adjeroh. A new algorithm for "the LCS problem" with application in compressing genome resequencing data. *BMC Genomics*, 17(S4), August 2016. 14

[BD21] Karl Bringmann and Debarati Das. A Linear-Time $n^{\{}0.4\}$-Approximation for Longest Common Subsequence. In Nikhil Bansal, Emanuela Merelli, and James Worrell, editors, *48th International Colloquium on Automata, Languages, and Programming (ICALP 2021)*, volume 198 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 39:1–39:20, Dagstuhl, Germany, 2021. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. 12

[BEG+18] Mahdi Boroujeni, Soheil Ehsani, Mohammad Ghodsi, Mohammad Taghi Hajiaghayi, and Saeed Seddighin. Approximating edit distance in truly subquadratic time: Quantum and mapreduce. In Artur Czumaj, editor, *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2018, New Orleans, LA, USA, January 7-10, 2018*, pages 1170–1189. SIAM, 2018. 13

[BES06] Tugkan Batu, Funda Ergün, and Süleyman Cenk Sahinalp. Oblivious string embeddings and edit distance approximations. In *Proceedings of the Seventeenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2006, Miami, Florida, USA, January 22-26, 2006*, pages 792–801. ACM Press, 2006. 13

[BI18] Arturs Backurs and Piotr Indyk. Edit distance cannot be computed in strongly subquadratic time (unless SETH is false). *SIAM J. Comput.*, 47(3):1087–1097, 2018. 10, 13

[BJKK04] Ziv Bar-Yossef, T. S. Jayram, Robert Krauthgamer, and Ravi Kumar. Approximating edit distance efficiently. In *45th Symposium on Foundations of Computer Science (FOCS 2004), 17-19 October 2004, Rome, Italy, Proceedings*, pages 550–559. IEEE Computer Society, 2004. 13

[BK15] Karl Bringmann and Marvin Künnemann. Quadratic conditional lower bounds for string problems and dynamic time warping. In Venkatesan Guruswami, editor, *IEEE 56th Annual Symposium on Foundations of Computer Science, FOCS 2015, Berkeley, CA, USA, 17-20 October, 2015*, pages 79–97. IEEE Computer Society, 2015. 13

[BK18] Karl Bringmann and Marvin Künnemann. Multivariate fine-grained complexity of longest common subsequence. In *SODA*, 2018. 10, 13

[BR20] Joshua Brakensiek and Aviad Rubinstein. Constant-factor approximation of near-linear edit distance in near-linear time. In *Proceedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing*, STOC 2020, page 685–698, New York, NY, USA, 2020. Association for Computing Machinery. 13

[CDG+18] Diptarka Chakraborty, Debarati Das, Elazar Goldenberg, Michal Koucký, and Michael E. Saks. Approximating edit distance within constant factor in truly sub-quadratic time. In Mikkel Thorup, editor, *59th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2018, Paris, France, October 7-9, 2018*, pages 979–990. IEEE Computer Society, 2018. 13

[CGL+19] Lijie Chen, Shafi Goldwasser, Kaifeng Lyu, Guy N. Rothblum, and Aviad Rubinstein. Fine-grained complexity meets IP = PSPACE. In Timothy M. Chan, editor, *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2019, San Diego, California, USA, January 6-9, 2019*, pages 1–20. SIAM, 2019. 10, 14

[HSSS19] MohammadTaghi Hajiaghayi, Masoud Seddighin, Saeed Seddighin, and Xiaorui Sun. Approximating LCS in linear time: Beating the $\sqrt{n}$ barrier. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2019, San Diego, California, USA, January 6-9, 2019*, pages 1181–1200. SIAM, 2019. 12

[JMV18] Hamidreza Jahanjou, Eric Miles, and Emanuele Viola. Local reduction. *Inf. Comput.*, 261:281–295, 2018. 13

[KS20] Michal Koucký and Michael Saks. Constant factor approximations to edit distance on far input pairs in nearly linear time. In *Proceedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing*, STOC 2020, pages 699—-712, New York, NY, USA, 2020. Association for Computing Machinery. 13

[LMS98] Gad M. Landau, Eugene W. Myers, and Jeanette P. Schmidt. Incremental string comparison. *SIAM Journal on Computing*, 27(2):557–582, April 1998. 13

[MP80] William J. Masek and Mike Paterson. A faster algorithm computing string edit distances. *J. Comput. Syst. Sci.*, 20(1):18–31, 1980. 9

[Nav01] Gonzalo Navarro. A guided tour to approximate string matching. *ACM Comput. Surv.*, 33(1):31–88, March 2001. 14

[RS20] Aviad Rubinstein and Zhao Song. Reducing approximate longest common subsequence to approximate edit distance. In *Proceedings of the Thirty-First Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '20, page 1591–1600, USA, 2020. Society for Industrial and Applied Mathematics. 10, 12, 18, 20, 21, 22, 23, 25, 27, 29, 31, 39, 41

[RSSS19] A. Rubinstein, S. Seddighin, Z. Song, and X. Sun. Approximation algorithms for lcs and lis with truly improved running times. In *2019 IEEE 60th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 1121–1145, 2019. 12, 42

[RW19] Aviad Rubinstein and Virginia Vassilevska Williams. Seth vs approximation. *SIGACT News*, 50(4):57–76, December 2019. 13

[Wil13] Ryan Williams. Improving exhaustive search implies superpolynomial lower bounds. *SIAM Journal on Computing*, 42(3):1218–1244, January 2013. 13

[Wil14] Ryan Williams. Nonuniform acc circuit lower bounds. *J. ACM*, 61(1), January 2014. 13

[Wil15] Virginia Vassilevska Williams. Hardness of Easy Problems: Basing Hardness on Popular Conjectures such as the Strong Exponential Time Hypothesis (Invited Talk). In Thore Husfeldt and Iyad Kanj, editors, *10th International Symposium on Parameterized and Exact Computation (IPEC 2015)*, volume 43 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 17–29, Dagstuhl, Germany, 2015. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. 13